**Документација за проект по Вештачка Интелигенција**

**(Battleship AI)**

# Вовед во Проектот

Battleship е стратегиска игра за двајца играчи. Секој има табла (обично 10x10) и тајно поставува флота од бродови со различна должина (на пример: 5, 4, 3, 3, 2 полиња).

* Играчите наизменично погодуваат кажувајќи координата (пример: B7).
* Противникот одговара со **„погодок“** ако таму има брод, или **„промашување“** ако е празно поле.
* Кога сите полиња на бродот се погодени, бродот е **потопен**.
* Играта завршува кога некој ќе ги потопи сите бродови на противникот.

Целта е со **помалку потези** да ја пронајдеш и потопиш целата флота на противникот.

Овој проект е изработен во **Python** и целта е да се создадат три различни модели/агенти кои користат различни стратегии и различни пристапи. Истражувањето за играта ме доведе до неколку основни стратегии кои мораат сите модели да ги поседуваат, овие стратегии се непоходни за најефикасно играње и клучните разлики се наоѓаат во одлуката за која позиција да ја одбере.

За подобра визуелизација на агентите искористена е **Pygame** библиотеката со која убаво е преставена играта. Во оваа документација ќе ги поминеме трите модели кои ги имплементирав, нивната ефикасност и начинот на функционирање.

# Основи на моделите

Моделите поседуваат заеднички елементи главно во делот на самото воспоставување на играта, нејзината визуелизација и текот на операции. Тие ги делат истите променливи, исти функции кои се импоритраат и истиот „game loop“. Истотака сите модели имаат заеднички стратегии кои се неопходни за ефикасно погодување, разликата се наоѓа во погодување на полиња во ai\_turn() функцијата.

**Game state –** променливи кои ја следат состојбата на играта

A computer code with colorful text

AI-generated content may be incorrect.

**Utility/modules files** – За подобра организација на проектот заедничките функции се ставени во посебни фајлови и импортирани при потреба

**game\_utils.py** – Ги содржи функции потребни за функционирање на играта

**graphics\_utils.py** – Ги содржи функциите за визуелизирање на играта

**statistics\_utils.py** – Ги содржи функциите за следење на статистики

## Воспоставување на играта

По декларирањето на Game state променливите се генерираат бродовите на играчот и бродовите на спротивникот(AI Моделот) преку **generate\_ships()** фунцкијата. Се повикува **draw\_grid()** функцијата, се исцртуваат бродовите на играчот и спротивничкото поле(grid).

Во **while running:** циклус се одвива „game loop“ што преставува операциите кои се повторуваат се дур не заврши играта, тука прв потег има играчот па AI моделот, разликата во сите модели се наоѓа во ai\_turn() функцијата која има различна имплементација кај секој модел.

## Основни стратегии

Секој модел поседува основни стратегии кои се неопходни за најефикасно погодување.

**Checkers pattern –** Идејата е дека сите бродови во играта (2-, 3-, 4- или 5-должина) зафаќаат повеќе полиња по ред.  
Тоа значи дека **не може да постојат сите бродови само на „црните“ полиња или само на „белите“** ако си замислите шаховска табла.

**✅ Придобивки**

1. **Дупло помалку потези** – бидејќи гаѓаш само половина од таблата.
2. **Гарантирано покривање** – секој брод ќе биде погоден барем еднаш ако користиш checkerboard.
3. **Ефикасно пребарување** – кога ќе погодиш еден дел, потоа преминуваш во “target mode” (проверуваш околу тоа поле за да го најдеш целиот брод).

Hunt & Target – дво фазна стратегија каде што првин бараш брод, кога ќе го најдеш преминуваш во Target и пробуваш целосно да го потонеш.

**1️⃣ Hunt (ловење)**

* Цел: **да најдеш барем еден дел од брод.**
* Во оваа фаза обично се користи **checkerboard guessing** (шаховска шема) и определниот агент користи посебна стратегија.
* Се гаѓаат полиња по шаблон за да се минимизира бројот на потези, со гаранција дека секој брод ќе биде погоден барем еднаш.
* Додека нема погодок, си во "hunt mode".

**2️⃣ Target (таргетирање)**

* Кога ќе погодиш **едно поле со брод**, тогаш **менуваш стратегија**.
* Наместо да продолжиш со checkerboard, почнуваш да гаѓаш **околу погодокот (горе, долу, лево, десно)**.
* Целта е да откриеш во која насока е поставен бродот (хоризонтално или вертикално).
* Кога ќе ја најдеш насоката, продолжуваш да пукаш по тој ред/колона додека не го потопиш целиот брод.

**🔄 Како изгледа циклусот**

1. Почнуваш со **hunt** → checkerboard pattern + AI Model.
2. Погодок! → преминуваш во **target mode**.
3. Го уништуваш бродот.
4. Се враќаш пак во **hunt mode** за следниот брод.

# Типови модели/агенти

Создадов три посебни модели/агенти кои користат различни стратегии, тие се **heatmap, expectimax** и **monte carlo** моделот.

## Heatmap Agent

Heatmap е **попаметна верзија на Hunt and Target**. Наместо да гаѓаш случајно или по checkerboard, ја користиш логиката на распоредување на бродовите за да пресметаш **кои полиња се најверојатни да содржат брод**.

**🧠 Како работи**

1. **Генерираш heatmap** → за секое поле на таблата пресметуваш колку можни позиции на брод можат да поминат низ него.
   * На пример: ако имаш брод со должина 4, проверуваш колку начини може да се постави на таблата така што тој квадрат е дел од бродот.
   * Поголем број = поголема шанса таму да има брод.
2. **Избираш најтопло поле** → пукаш на квадратот со највисока вредност (најголема веројатност).
3. **Ажурираш heatmap после секој потег** → ако е промашување, тие можни позиции се отфрлаат; ако е погодок, тогаш фокусираш околу погодокот (слично на Target mode).

**✅ Предности**

* Многу поефикасна од случајно или checkerboard пукање.
* Се адаптира според тоа кои бродови останале и кои позиции веќе се невозможни.
* Намалува просечен број на потези за победа (AI со heatmap често победува за ~40 потези или помалку на 10x10 табла).

Со други зборови: **heatmap стратегијата е статистички ловец** – пукаш таму каде што е најверојатно да има брод, наместо да гаѓаш на слепо.

## Expectimax Agent

Супер, ајде истото објаснување за твојот **Expectimax Battleship AI**, ама на македонски 👇

**🧠 Што прави овој AI**

AI-то користи **Expectimax** за да одлучи на кое поле да пука. Наместо само да користи heatmap или случајно да гаѓа, тој симулира и:

1. **Hunt mode (ловење)** – користи expectimax со heatmap за да бара нови бродови.
2. **Target mode (таргетирање)** – ако има веќе погодени полиња, пробува да го потопи бродот со enhanced\_target\_shot\_multi\_ship.

**🔑 Главни делови од кодот**

**1. Heatmap основа**

* def compute\_heatmap(occupied, remaining\_ships):
* heatmap = create\_board()
* for ship in remaining\_ships:
* mark\_ship\_positions(heatmap, occupied, len(ship))
* return heatmap
* Истото како **heatmap стратегија**: пресметува каде може да бидат бродовите врз основа на тоа што е веќе познато (hits/misses).
* Поголема вредност = поголема веројатност таму да има брод.

**2. Хеуристика**

* def heuristic\_value(occupied, current\_hits, enemy\_hits, enemy\_misses):
* return len(enemy\_hits) - 0.2\*len(enemy\_misses)
* Оценка: повеќе погодоци = добро, повеќе промашувања = малку лошо.
* Се користи кога се стигнува до крај на пребарување (max depth).

**3. Expectimax јадро**

* def expectimax(...):
* if depth == 0 or all\_ships\_sunk(...):
* return heuristic\_value(...), None
* Ако сме на максимална длабочина или сите бродови се потопени → враќа хеуристика.

**4. Генерирање кандидати**

* candidates = [(r,c,heatmap[r][c]) for ... if occupied[r][c]==0]
* candidates.sort(key=lambda x: x[2], reverse=True)
* candidates = candidates[:TOP\_K]
* Ги зема сите празни полиња.
* Ги сортира според вредност од heatmap.
* Ги задржува само **TOP K најдобри потези**.

**5. Expectation чекор**

* p\_hit = val / max\_heat
* ...
* exp\_val = p\_hit \* (1 + GAMMA\*val\_hit) + (1-p\_hit) \* (GAMMA\*val\_miss)
* За секој кандидат (r,c):
  + Пресметува веројатност за погодок (p\_hit).
  + Симулира гранка **hit**.
  + Симулира гранка **miss**.
  + Ги комбинира преку **очекувана вредност**.
  + GAMMA е фактор за „идни потези“ (discount).

Ова е суштината на **Expectimax** → не претпоставува најлош случај (како Minimax), туку просек врз база на веројатности.

**6. Избор на најдобар потег**

* if exp\_val > best\_val:
* best\_val = exp\_val
* best\_move = (r,c)
* Го бира потегот со највисока очекувана вредност.

**7. Логика за потег на AI**

* if not current\_hits:
* mode = "hunt"
* else:
* mode = "target"
* Ако нема погодени полиња → оди во **hunt mode** (Expectimax).
* Ако има погодени полиња → оди во **target mode** за да го потопи бродот.

**🔄 Како сè се поврзува**

1. **Heatmap** дава веројатност каде може да бидат бродовите.
2. **Expectimax** користи тие веројатности и симулира повеќе потези напред (hit/miss гранки).
3. AI го избира потегот што носи најголем **очекуван резултат**.

**✅ Зошто е силно**

* Подобро од обичен heatmap → затоа што гледа неколку чекори напред.
* Подобро од Minimax → затоа што средината е случајна (не е секогаш најлош исход).
* Комбинира **веројатност (heatmap)** + **планирање (expectimax)**.

## Monte Carlo Agent

**🧠 Што прави овој AI**

AI-то користи **Monte Carlo симулации** за да одреди каде најверојатно се кријат бродовите:

* **Target mode** – ако има погодено брод, продолжува систематски да гаѓа околу погодените полиња за да го потопи.
* **Hunt mode** – ако нема активни погодоци, користи **checkerboard** + **Monte Carlo симулации** за да избере каде да пука.

**🔑 Главни делови**

**1. Monte Carlo потег**

* def monte\_carlo\_ai\_turn(...):
* if not current\_hits:
* mode = "hunt"
* else:
* mode = "target"
* Ако нема активни погодоци → **hunt mode**.
* Ако има → **target mode** со enhanced\_target\_shot\_multi\_ship.

**2. Target mode**

* shot, updated\_current\_hits = enhanced\_target\_shot\_multi\_ship(...)
* Прво пробува паметно таргетирање околу познатите погодоци.
* Ако нема повеќе валидни соседи → се враќа на **hunt mode**.

**3. Hunt mode + Checkerboard**

* if (r + c) % 2 == 0:
* checkerboard\_shots.append((r, c))
* Наместо да ги проверува сите полиња, AI пука само на **црни полиња на шаховска табла**.
* Ова е оптимизација: бидејќи сите бродови се најмалку 2 полиња долги, нема потреба да проверува „бели“ полиња.

**4. Monte Carlo симулации**

* valid\_configs = []
* for \_ in range(min(simulations // 2, 10)):
* config = generate\_random\_ship\_configuration(...)
* if config:
* valid\_configs.append(config)
* Генерира неколку **случајни валидни конфигурации** на бродови кои се согласуваат со:
  + Постоечки **погодоци**.
  + Постоечки **промашувања**.
* Секоја конфигурација е еден можен „layout“ на непријателските бродови.

**5. Проценка на потези**

* for shot in shots\_to\_evaluate:
* hits = sum(1 for config in valid\_configs if shot in config)
* shot\_scores[shot] = hits / len(valid\_configs)
* За секој можен потег (checkerboard полиња):
  + Го проверува колку често се појавува брод таму во симулираните конфигурации.
  + Тоа станува **веројатност за погодок**.

**6. Избор на потег**

* best\_shot = max(shot\_scores.keys(), key=lambda s: shot\_scores[s])
* AI го бира потегот со **највисока веројатност за погодок**.
* Ако не успее да генерира конфигурации → избира случаен checkerboard потег (fallback).

**7. Execute Shot**

* if (r, c) in player\_coords:
* enemy\_hits.add((r, c))
* current\_hits.append((r, c))
* Проверува дали погодил.
* Ако погодил → додава во enemy\_hits и current\_hits.
* Ако цел брод е потопен → печати debug порака.

**8. Debug & статистика**

* Секој потег печати дали е checkerboard или обичен.
* Се води статистика:
  + Број на потези на AI.
  + Просечен број на потези при победа.

**🔄 Како сè се поврзува**

1. **Target mode** → паметно завршува потопување.
2. **Hunt mode** → користи checkerboard за да намали простор.
3. **Monte Carlo** → симулира повеќе можни распореди и пресметува веројатности за погодоци.
4. Избира најдобар потег базиран на статистички резултат.